JKU
LINZ INSTITUTE
OF TECHNOLOGY

PARALLEL COMPUTING
Algorithms and Complexity

Univ.-Prof. Dr. Alois Zoitl
LIT | Cyber-Physical Systems Lab
Johannes Kepler University Linz

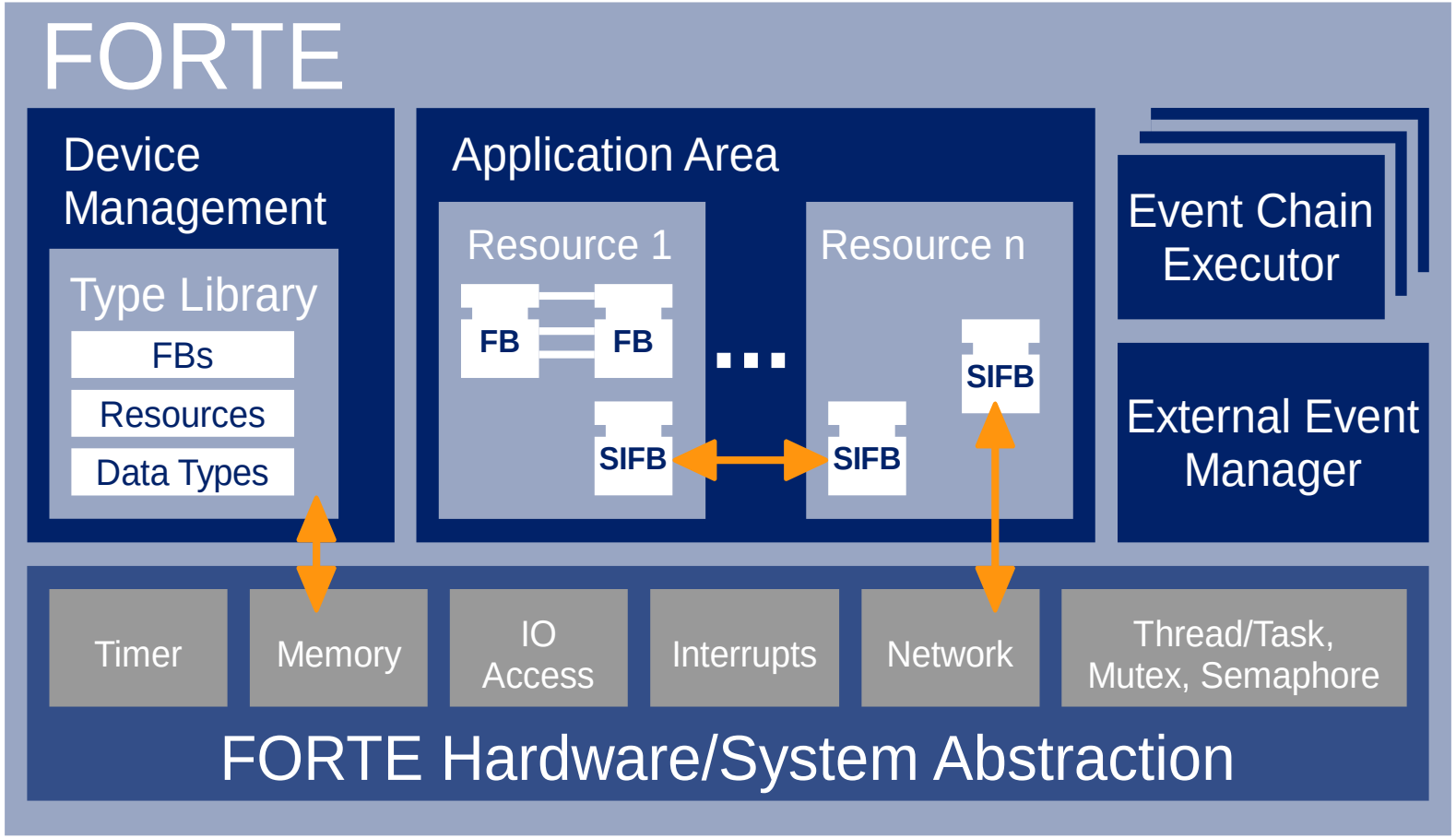
To whom honor is due....

These slides are based on a slide deck from

Prof. Dr. Armin Biere

from whom I took over this lecture.
He deserves thanks for his kind permission to use them.

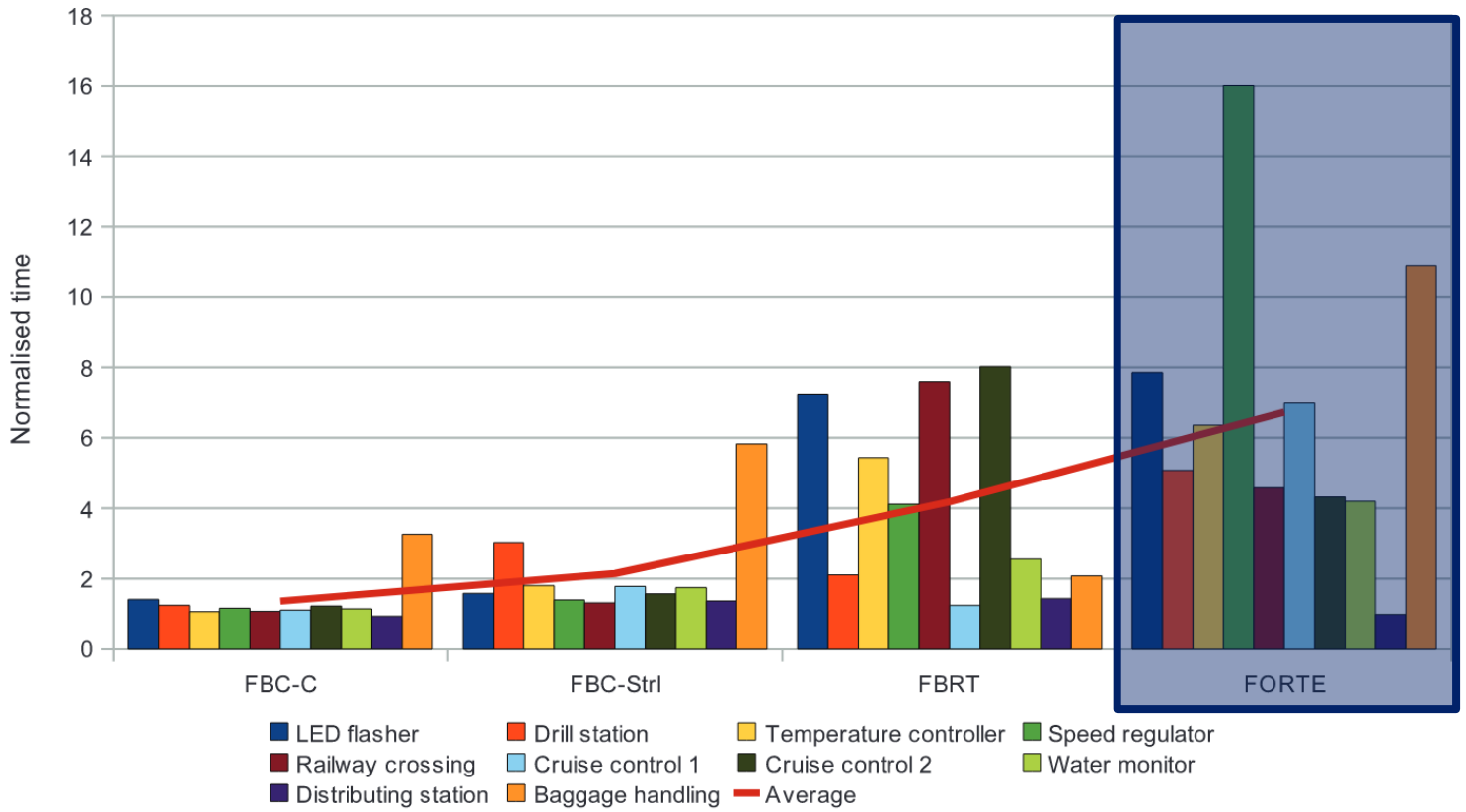
My Background – Embedded Real-time Computing



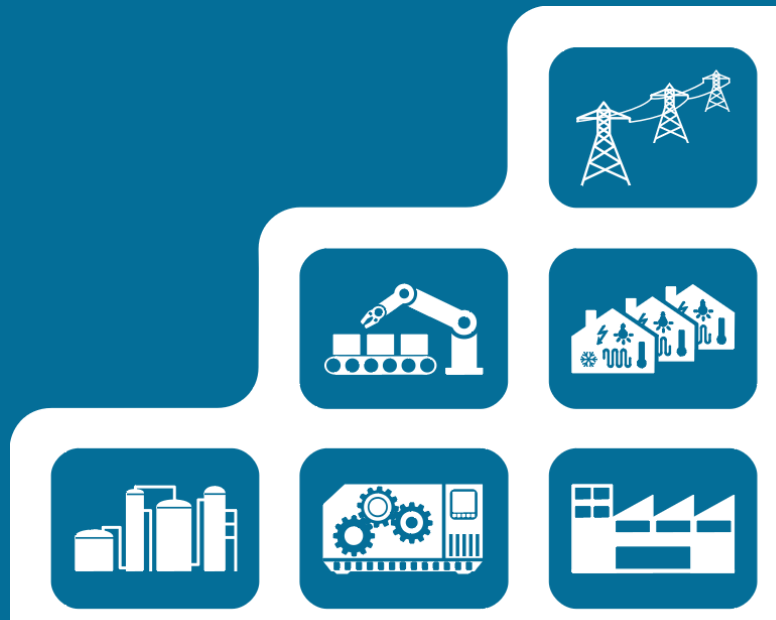
Eclipse 4diac: https://www.eclipse.dev/4diac

Synchronisation Penalty

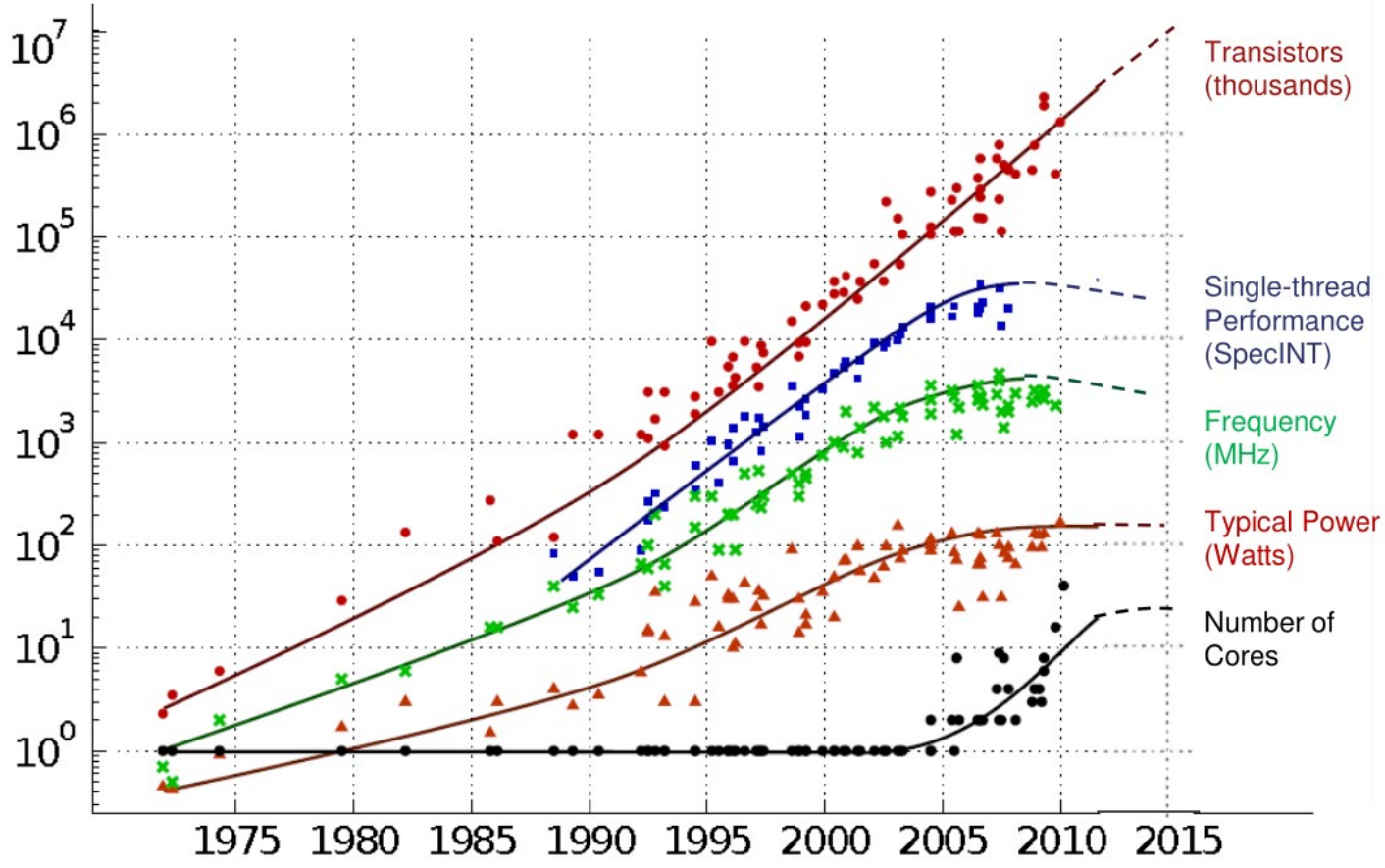
Fig. 10 of
Implementing Constrained Cyber-Physical Systems with IEC 61499.
Yoong, Roop, and Salcic,
http://dx.doi.org/10.1145/2362336.2362345



Need for Parallelization:
End of Moores Law on Single Core



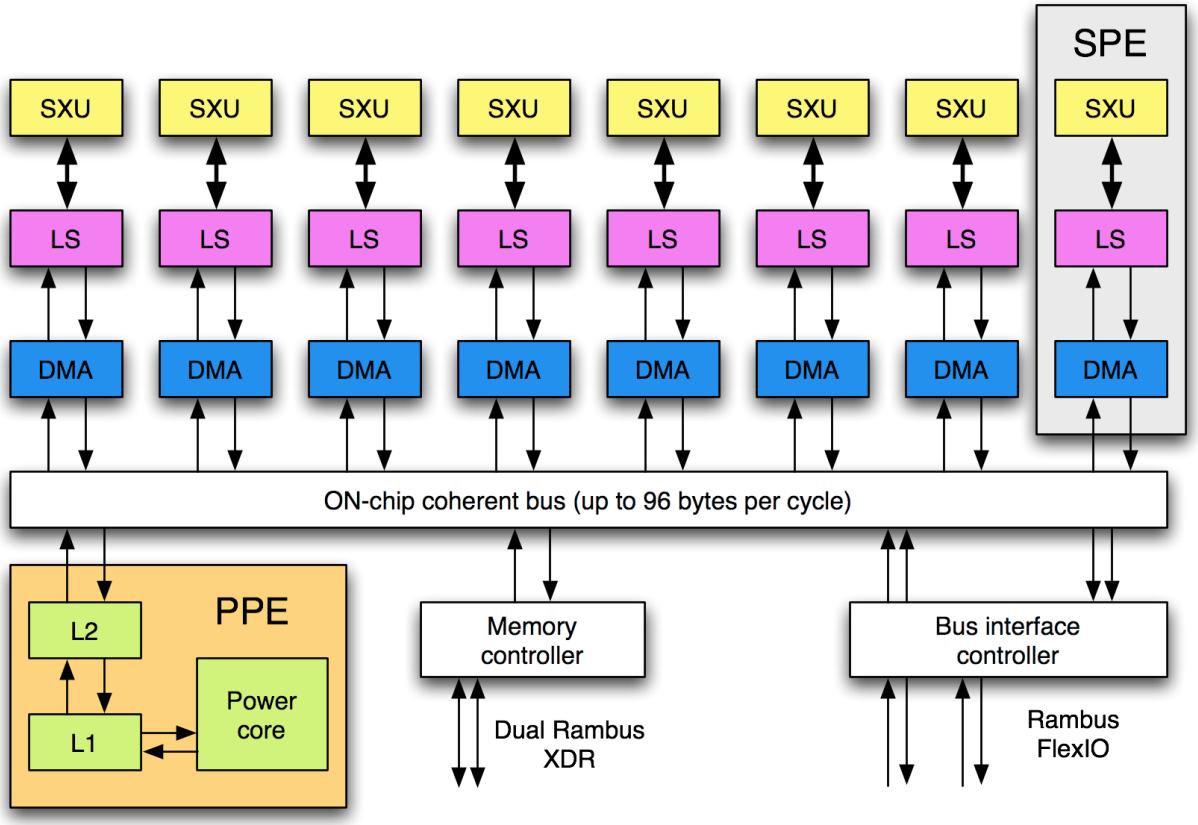
35 Years of Microprocessor Trend Data



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Source: Chuck Moore, Data Processing in Exascale-Class Computer Systems, 2011

Playstation 3: Cell Processor



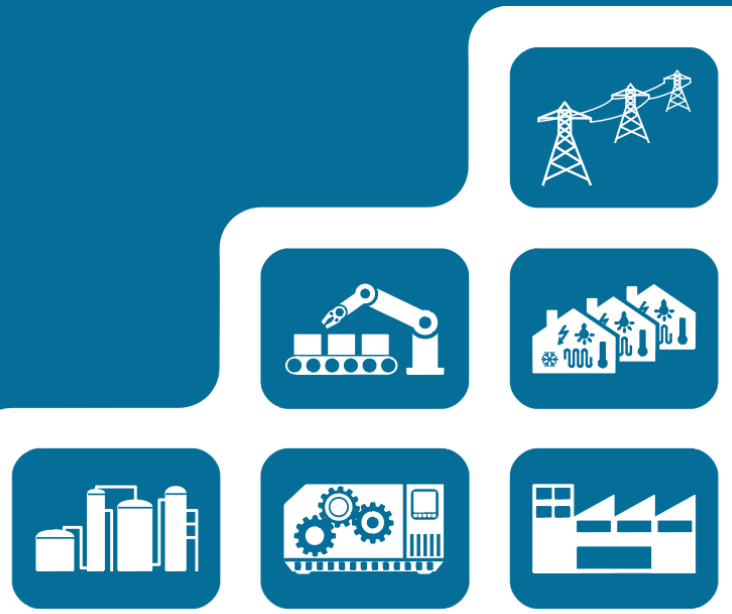
Source: Wipedia

Source: US Department of Defense

Overview Current CPU Architetures

- Intel
  - i5: 6 P / 8 E
  - i7: 8 P / 12 E
  - i9: 8 P / 16 E
- AMD
  - Ryzen: 4 – 96
- Raspberry Pi
  - Since Mod 2: 4
- Apple
  - M1: 4 – 16 P / 4 E
  - M2: 4 – 16 P / 4 – 8 E
  - M3: 4 – 12 P / 4 E
  - M4: 4 – 12 P / 4 – 10 E / 10 – 40 GPU
- Nvidia
  - Tesla: 128 – 18,176 Cuda cores

Parallelizing Existing Algorithms



Slow-Down in Parallel SAT

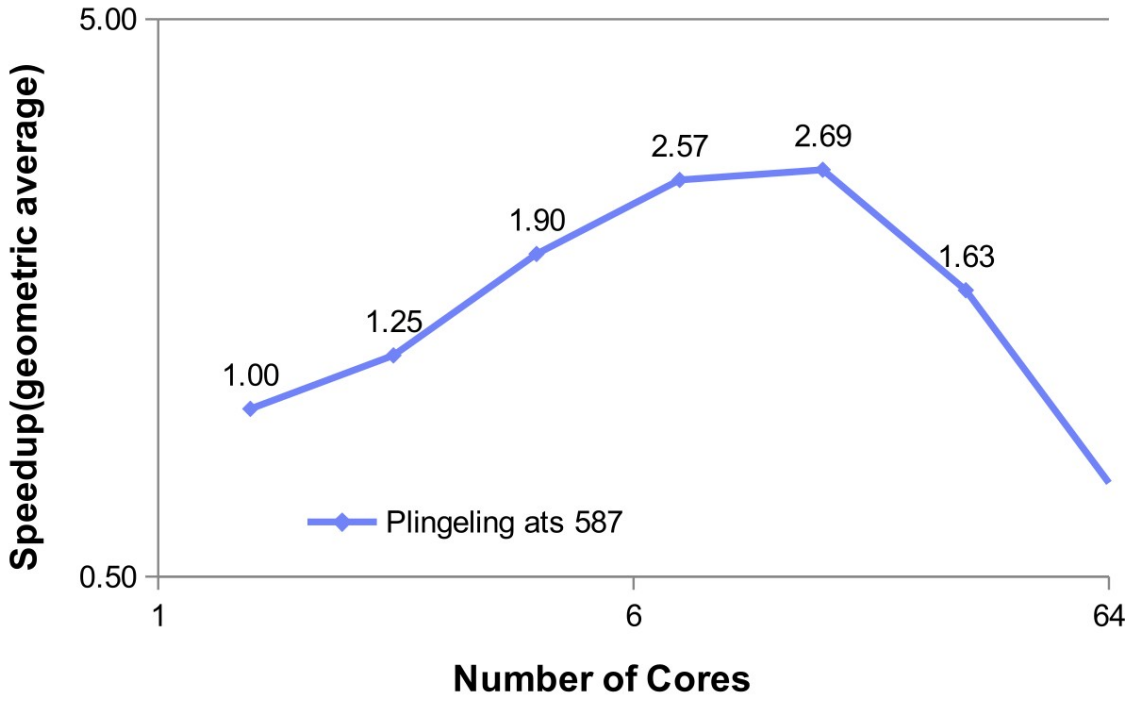
- Parallel Multithreaded Satisfiability Solver: Design and Implementation.
  Yulik Feldman, Nachum Dershowitz, Ziyad Hanna
  http://dx.doi.org/10.1016/j.entcs.2004.10.020
- Paper is inconclusive about the reason for slow-down
- Probably more threads work on useless sub-tasks
- Sharing clauses caching sub-computation increases pressure on memory system
- Maybe search space splitting was not a good idea (guiding path)

Table 2
Performance of SAT solver with different numbers of working threads

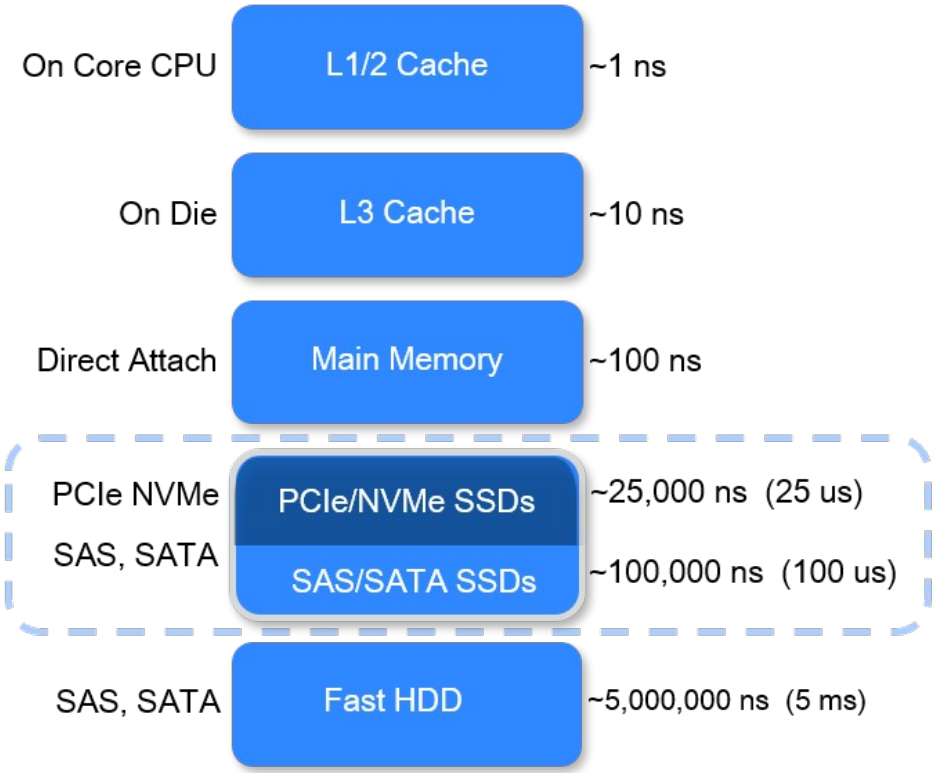
Configuration	One	Two	Three	Four	Four:One
A	13	15	61	89	6.8
B	20	21	42	47	2.4
C	14	16	19	22	1.6
D	13	15	14	15	1.2
E	7	7	7	10	1.4
F	8	20	27	53	6.6
G	6	55	195	168	28.0
H	6	52	86	107	17.8

Low Speedup in Parallel SAT

- http://www.birs.ca/events/2014/5-day-workshops/14w5101/videos/watch/201401221154-Sabharwal.html slide 4 of (video 3:30)
- Sequential SAT algorithms produce proofs of large depth (= span)
- So need new algorithms which produce low depth proofs



Limiting Factor Memory Access?



Memory System is Good Enough

- Analysis of Portfolio-Style Parallel SAT Solving on Current Multi-Core Architectures.
  Martin Aigner, Armin Biere, Christoph Kirsch, Aina Niemetz, Mathias Preiner.
  http://fmv.jku.at/papers/AignerBiereKirsch NiemetzPreiner-POS13.pdf
- Largest speed-up obtained by portfolio approach
  - Run different search strategies in parallel
  - If one terminates stop all
  - In practice share some important learned clauses caching sub-computations
- Slow-down due to memory system?
  - Since memory system (memory / caches / bus) are shared in multi-core systems
  - Slow-down not too bad (particularly for solvers with small working set)
  - Even though considered memory-bound (but random access)
  - Waiting time for memory to arrive overlaps

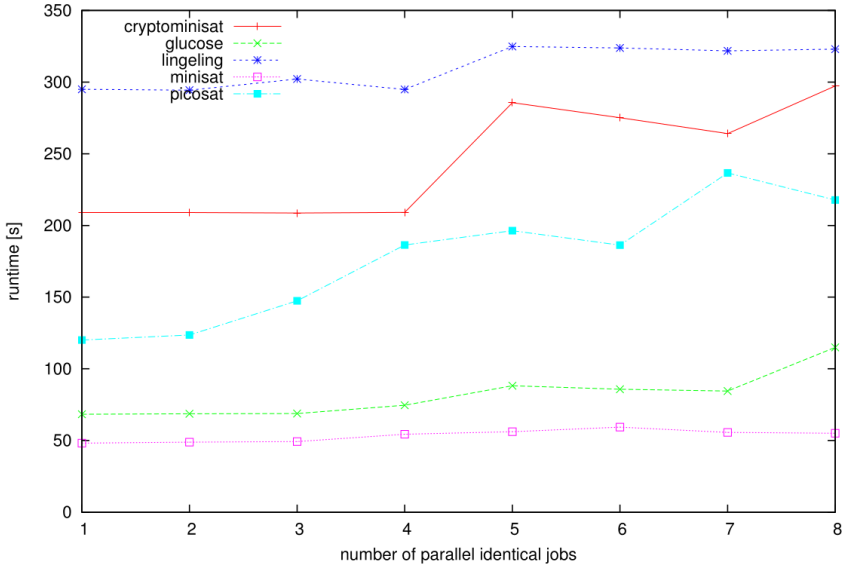


Figure 8: Absolute runtime required for an increasing number of parallel jobs solving the narain-vpn-clauses-10 benchmark on the amd-opteron-2350-8vcores machine.

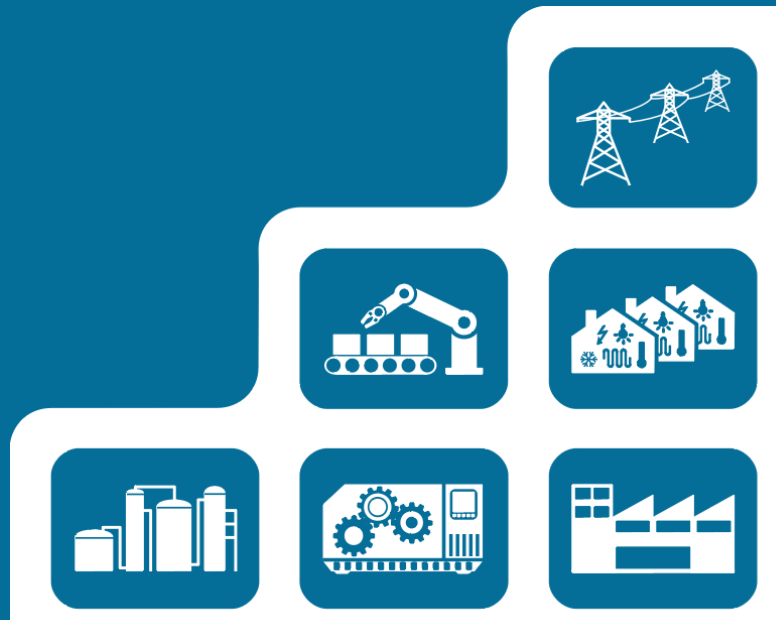
Clever Splitting

- Marijn Heule, Oliver Kullmann, Siert Wieringa, Armin Biere.
  Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads.
  Haifa Verification Conference 2011: 50-65, Springer 2012
  http://dx.doi.org/10.1007/978-3-642-34188-5_8

- Marijn J.H. Heule, Oliver Kullmann, and Victor Marek
  Solving and Verifying the boolean Pythagorean Triples problem via Cube-and-Conquer.
  SAT 2016, 196-211, Springer 2016
  http://dx.doi.org/10.1007/978-3-319-40970-2_15

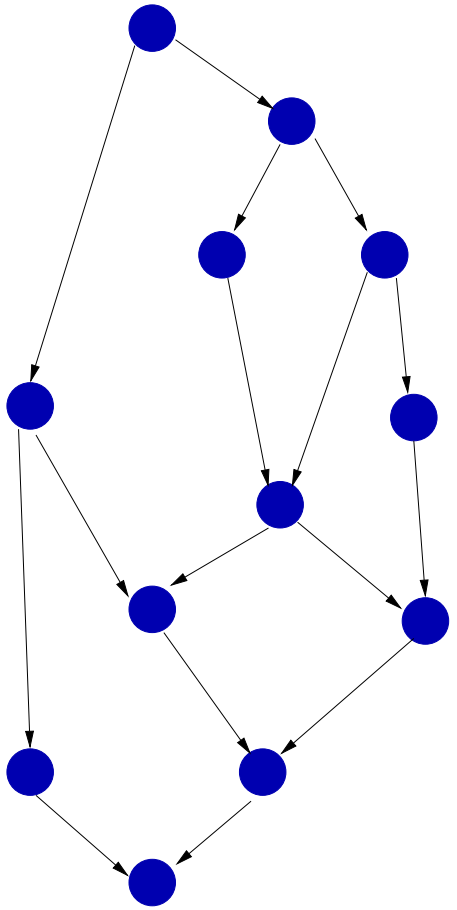
- Everything is Bigger in Texas
  https://www.cs.utexas.edu/~marijn/ptn/
  JKU CS Colloquium 22. June 2016

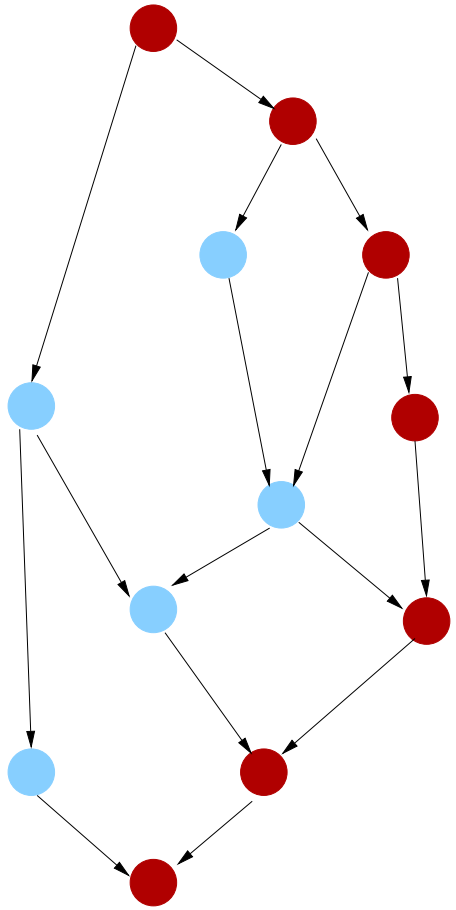
Theory on Parallelizabilty



Work and Span

work
12





span
7

Amdahl's Law with Work and Span

T = work = sequential time T_p = wall-clock time p CPUs T_∞ = wall-clock time ∞ CPUs

- Speedup: $S_P = T / T_P$
- *Span …* critical path (also called "makespan" in the context of scheduling)
- *f* … fraction of sequential work, thus

  f = span / work

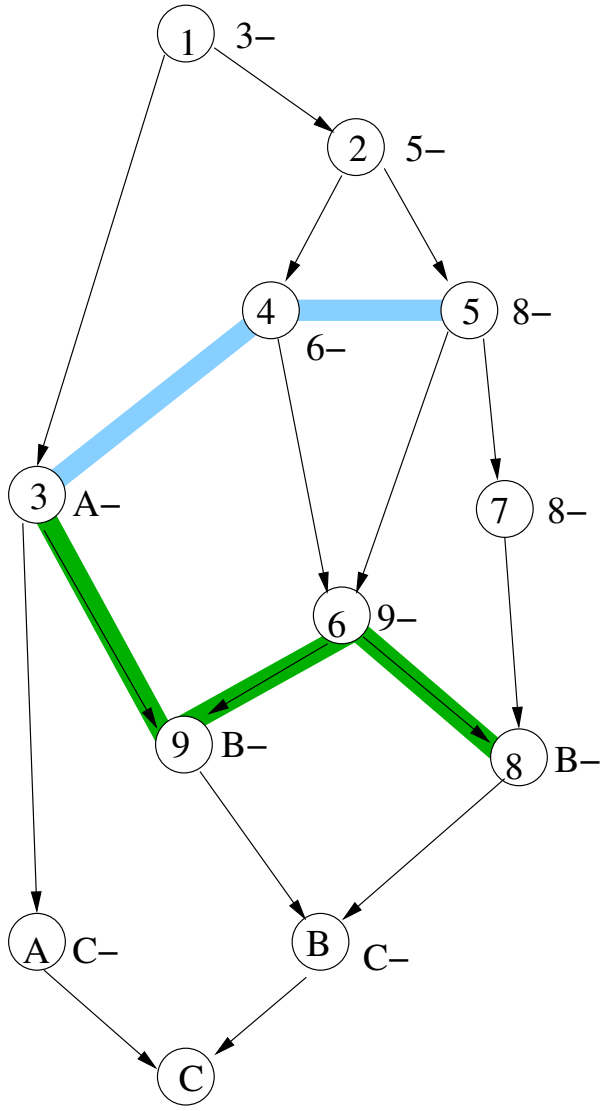
  Simplified Amdahl's law in terms of work and span: $S_p \leq 1/f$ = work /span

- Reduce *span* as much as possible:
  - ☐ keep sequential blocks short! → coarse grained locking is evil
  - ☐ keep sequential dependencies short! → (non-logarithmic) loops are evil

Pebble Games

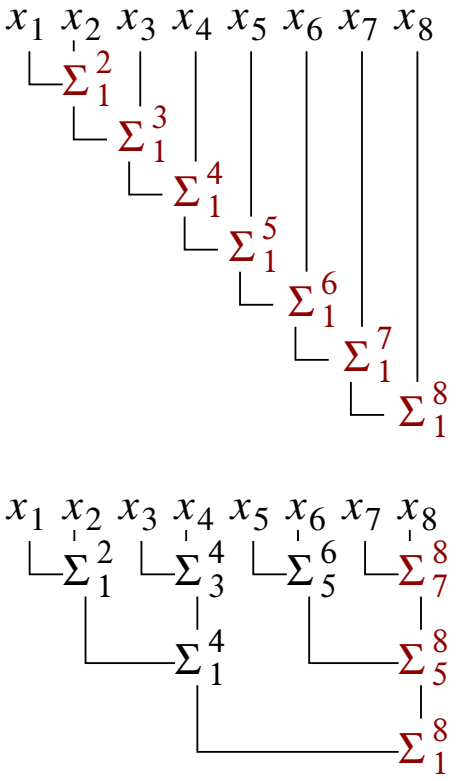
- Given a directed acyclic graph with one sink.
- Nodes of the graph have a pebble or not.
- One step can either . . .
  - . . . remove a pebble from a node . . .
  - . . . or add a new pebble to a node without one, . . .
  - . . . but only if all its predecessor have a pebble.
- Goal is to only have a pebble on the sink node.
- What is the smallest maximum number of pebbles needed?

- Common concept in complexity theory
  - Assuming intermediate results have to be stored
  - Relates to smallest p needed to reach maximum speed-up
  - This version (black pebble game) actually only gives space bounds



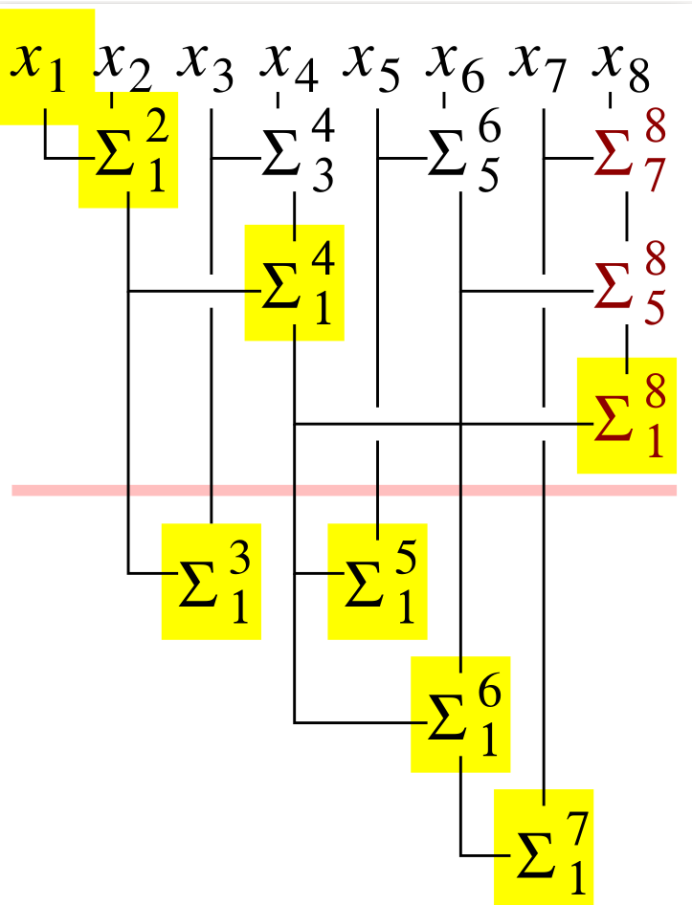
Sum

- Compute sum $\sum_{1}^{n} x_i$ for *n* numbers xi in parallel
- Sequential
  - $y_0 = 0$, $y_{i+1} = y_i + x_i$ for $i = 1 \ldots n - 1$
  - *work* = $T = O(n)$ (n − 1 additions)
  - *span* = $O(n)$ too
  - Since y_{i+1} depends on all previous y_j with $j \leq i$
  - Thus no speed-up Sp = O(1)
- Parallel
  - Associativity allows to regroup computation
  - Work = O(n) remains the same
  - Span = O(log n) reduces exponentially
  - Speed-up not ideal but S_n = O(n / log n)
  - Note p > n does not make sense



Prefix / Scan

- Compute all sums $s_j = \sum_{1}^{j} x_i$ for all $j = 1 \ldots n$ and again n numbers x_i in parallel
- Sequential version as in previous slide
- Parallel version needs a second depth *O(log n)* pass
- Works even “in place” (first pass overwrites original x_i)
- But actual “wiring” complicated
- Still *span = O(log n)*
- Basic algorithmic idea for many “parallel” algorithms
- Propagate and generate adders with prefix trees instead of ripple carry adder

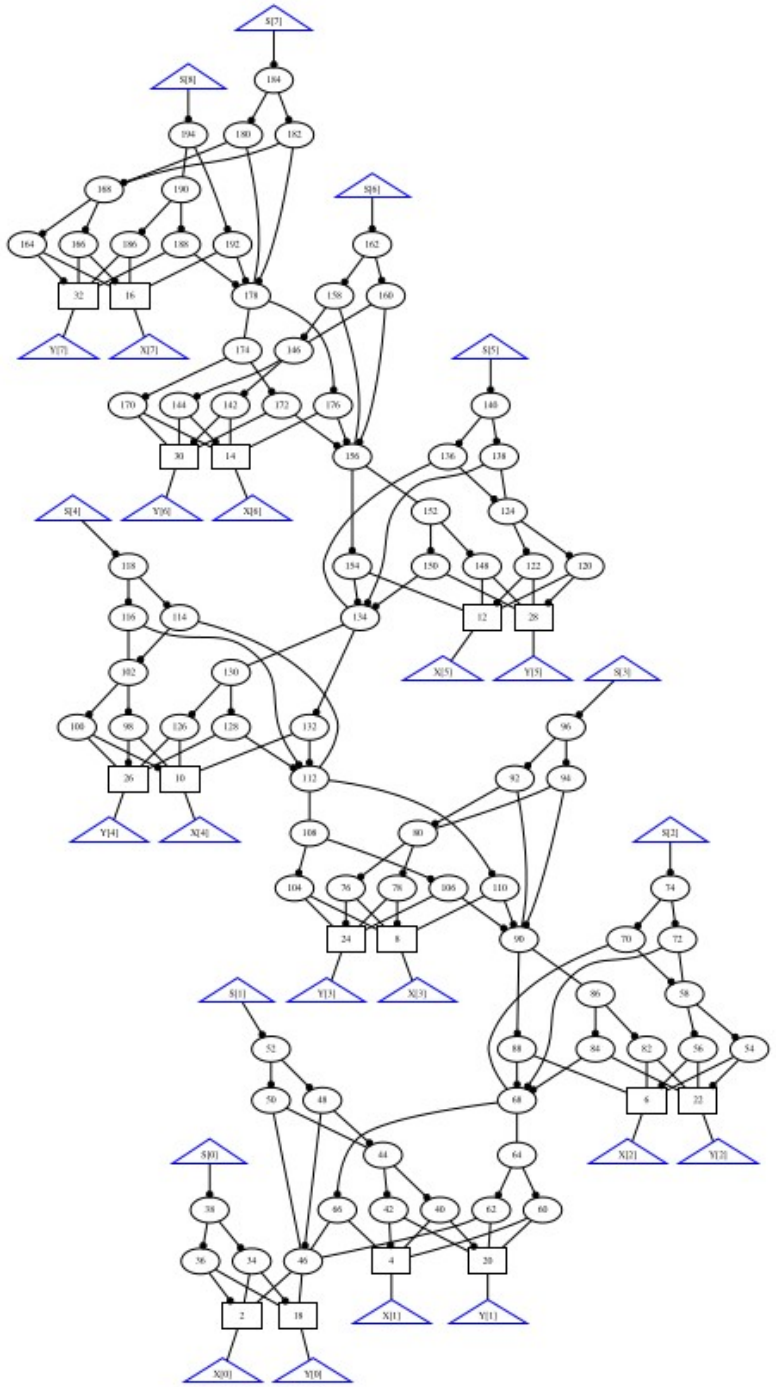


Ripple-Carry-Adder

$$
\begin{aligned}
s_i &= x_i \oplus y_i \oplus c_i && \text{sum} \\
c_{i+1} &= x_i\, y_i + x_i\, c_i + y_i\, c_i && \text{carry}
\end{aligned}
$$

$$
\begin{aligned}
& & c_0 &= 0 \\
s_0 &= x_0 \oplus y_0 & c_1 &= x_0\, y_0 \\
s_1 &= x_1 \oplus y_1 \oplus c_1 & c_2 &= x_1\, y_1 + x_1\, c_1 + y_1\, c_1 \\
s_2 &= x_2 \oplus y_2 \oplus c_2 & c_3 &= x_2\, y_2 + x_2\, c_2 + y_2\, c_2 \\
s_3 &= x_3 \oplus y_3 \oplus c_3 & c_4 &= x_3\, y_3 + x_3\, c_3 + y_3\, c_3 \\
s_4 &= x_4 \oplus y_4 \oplus c_4 & c_5 &= x_4\, y_4 + x_4\, c_4 + y_4\, c_4 \\
s_5 &= x_5 \oplus y_5 \oplus c_5 & c_6 &= x_5\, y_5 + x_5\, c_5 + y_5\, c_5 \\
s_6 &= x_6 \oplus y_6 \oplus c_6 & c_7 &= x_6\, y_6 + x_6\, c_6 + y_6\, c_6 \\
s_7 &= x_7 \oplus y_7 \oplus c_7 & c_8 &= x_7\, y_7 + x_7\, c_7 + y_7\, c_7
\end{aligned}
$$

$$work = \mathcal{O}(n) \qquad span = \mathcal{O}(n)$$



Propagate-and-Generate Adder / Lookahead Adder

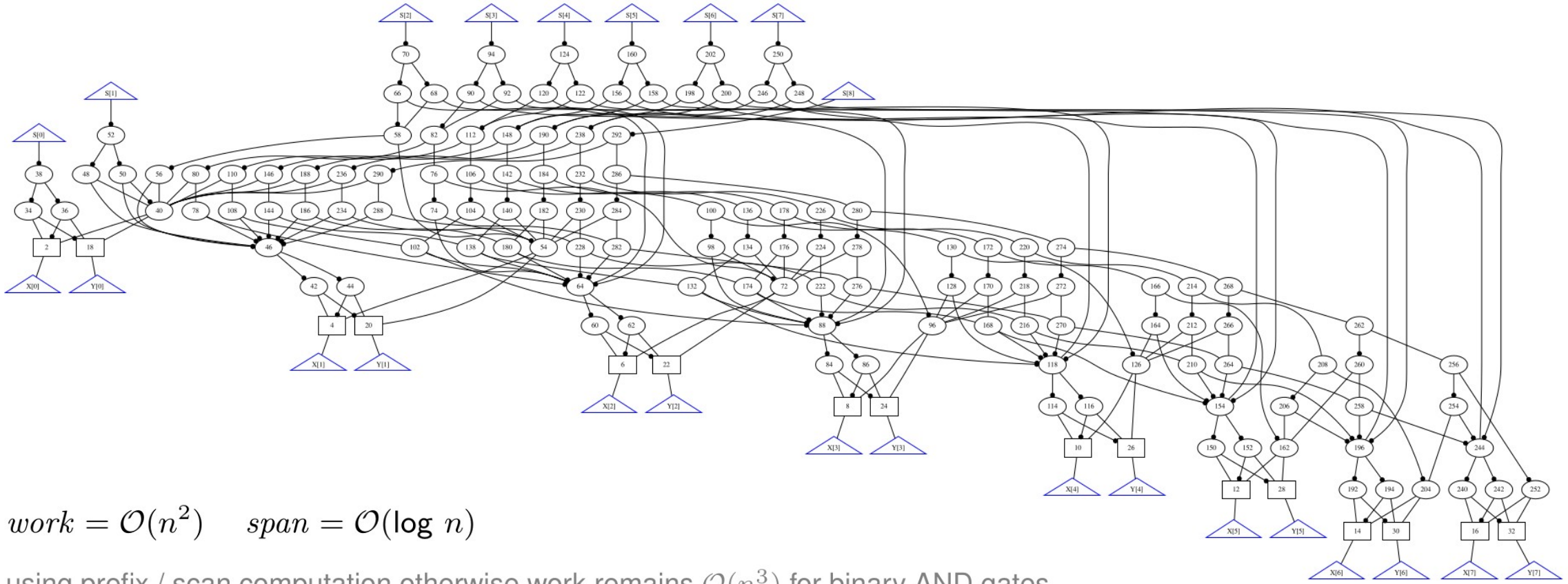
$$p_i = x_i + y_i \quad \text{propagate}$$
$$g_i = x_i\, y_i \quad \text{generate}$$

$$c_{i+1} = g_i + p_i\, c_i \quad \text{new carry computation formula}$$

$$\begin{aligned}
c_0 &= 0 \\
c_1 &= g_0 \\
c_2 &= g_1 + p_1\, g_0 \\
c_3 &= g_2 + p_2\, g_1 + p_2\, p_1\, g_0 \\
c_4 &= g_3 + p_3\, g_2 + p_3\, p_2\, g_1 + p_3\, p_2\, p_1\, g_0 \\
c_5 &= g_4 + p_4\, g_3 + p_4\, p_3\, g_2 + p_4\, p_3\, p_2\, g_1 + p_4\, p_3\, p_2\, p_1\, g_0 \\
c_6 &= g_5 + p_5\, g_4 + p_5\, p_4\, g_3 + p_5\, p_4\, p_3\, g_2 + p_5\, p_4\, p_3\, p_2\, g_1 + p_5\, p_4\, p_3\, p_2\, p_1\, g_0 \\
c_7 &= g_6 + \ldots + \ldots + p_6\, p_5\, p_4\, p_3\, p_2\, p_1\, g_0 \\
c_8 &= g_7 + \ldots + \ldots + p_7\, p_6\, p_5\, p_4\, p_3\, p_2\, p_1\, g_0
\end{aligned}$$

$work = \mathcal{O}(n^2)$ $span = \mathcal{O}(\log\ n)$ assuming n-ary gates otherwise $work = \mathcal{O}(n^3)$

Carry-Look-Ahead Adder

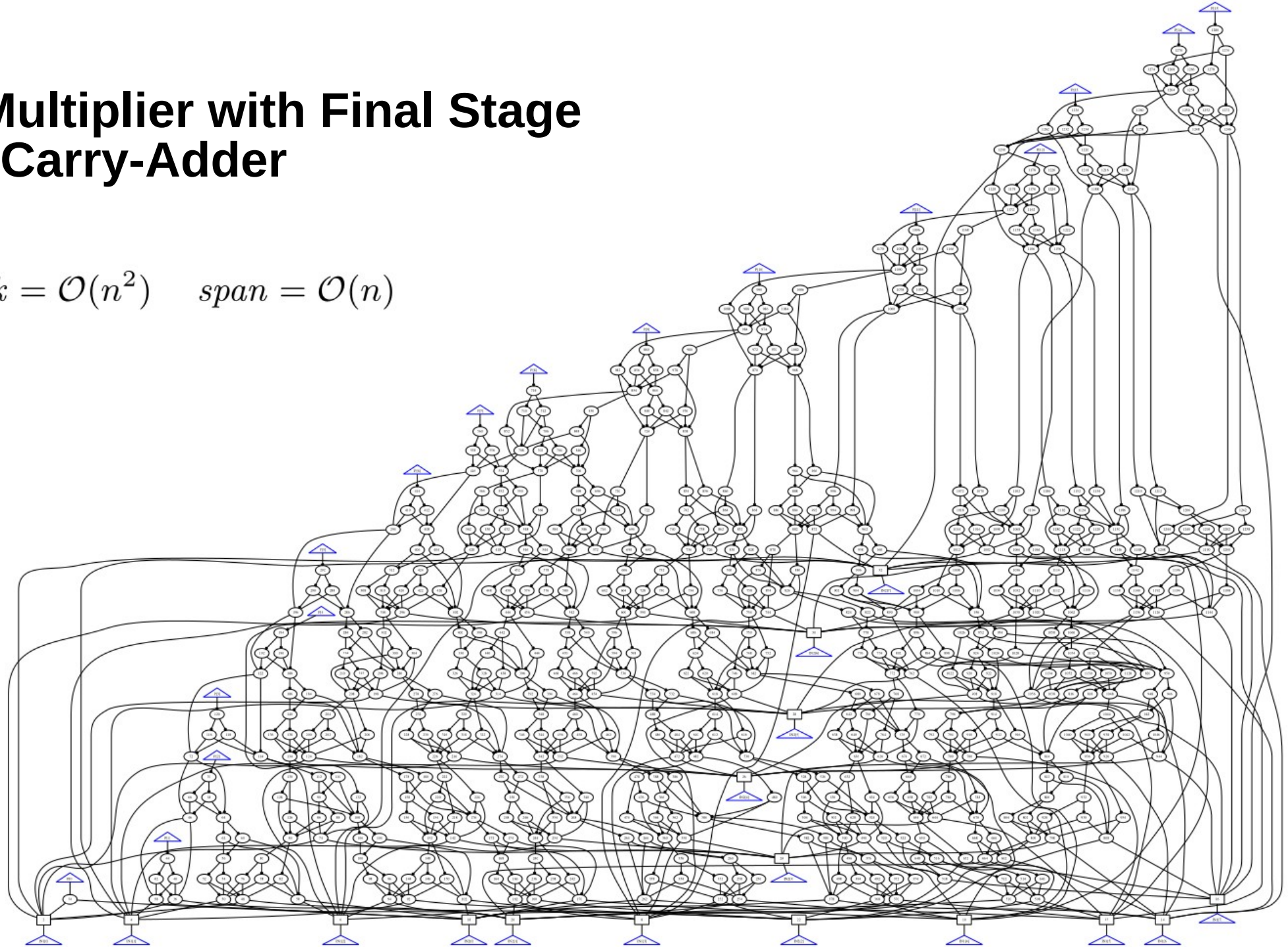


$work = \mathcal{O}(n^2) \quad span = \mathcal{O}(\log\ n)$

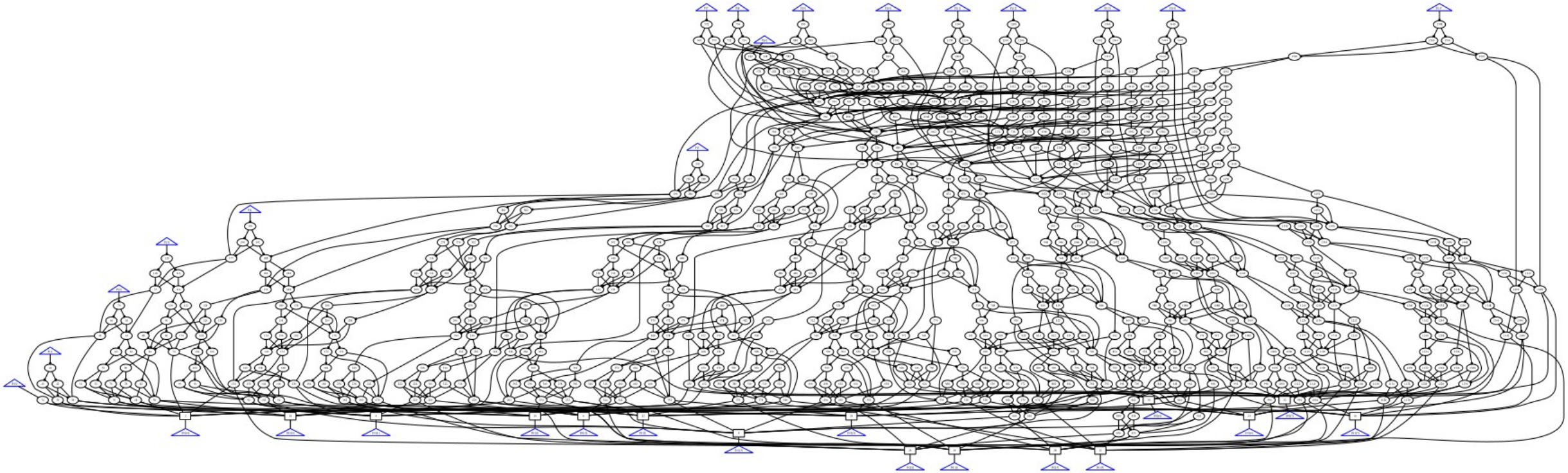
using prefix / scan computation otherwise work remains $\mathcal{O}(n^3)$ for binary AND gates

Array Multiplier with Final Stage Ripple-Carry-Adder

$$work = \mathcal{O}(n^2) \qquad span = \mathcal{O}(n)$$

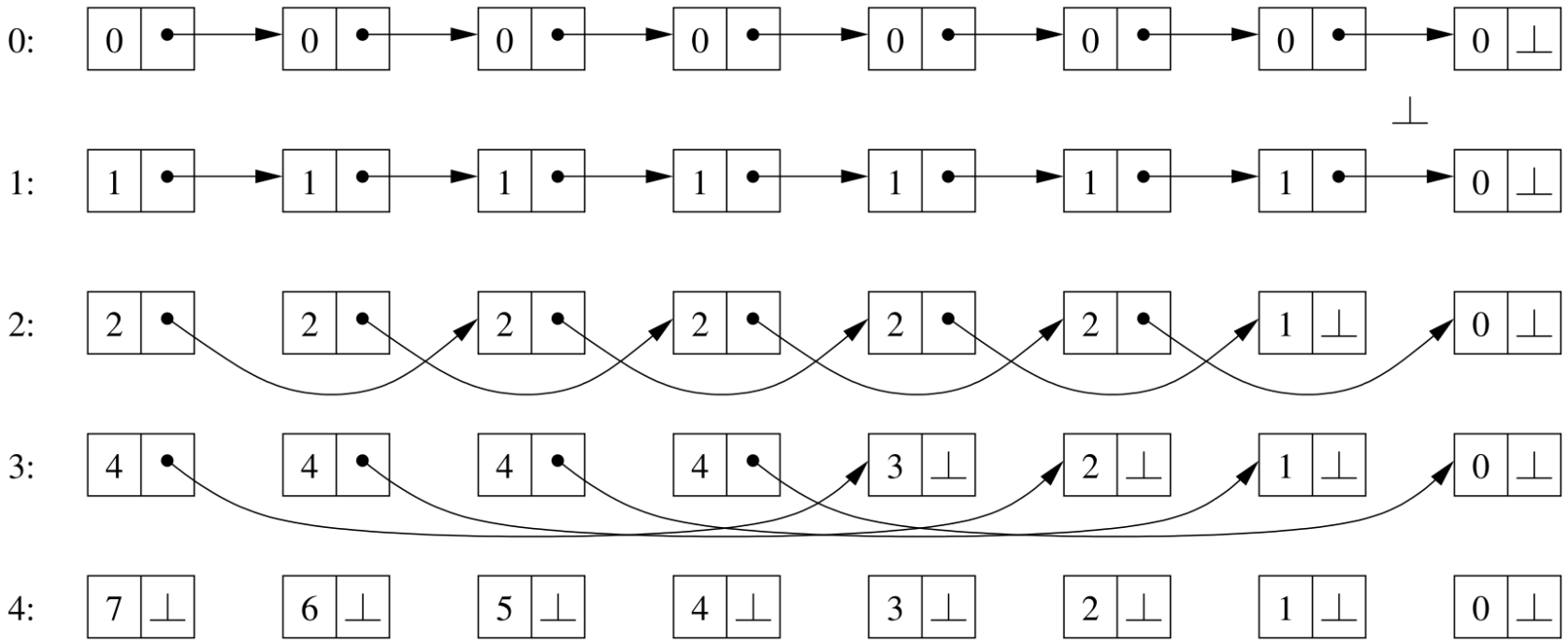


Wallace-Tree Multiplier with Final Stage Carry-Look-Ahead Adder



$$work = \mathcal{O}(n^2) \qquad span = \mathcal{O}(\log\ n)$$

List Ranking / Pointer Jumping



determine distance to head of list:

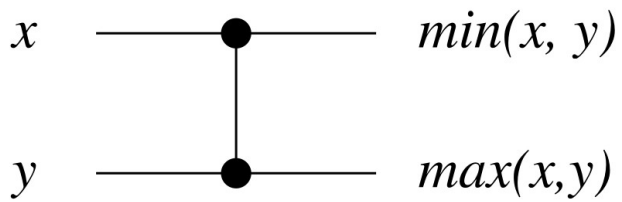
as long there is i with $next[i] \neq \perp$:

$val[i]$ += $val[next[i]]$

$next[i]$ = $next[next[i]]$

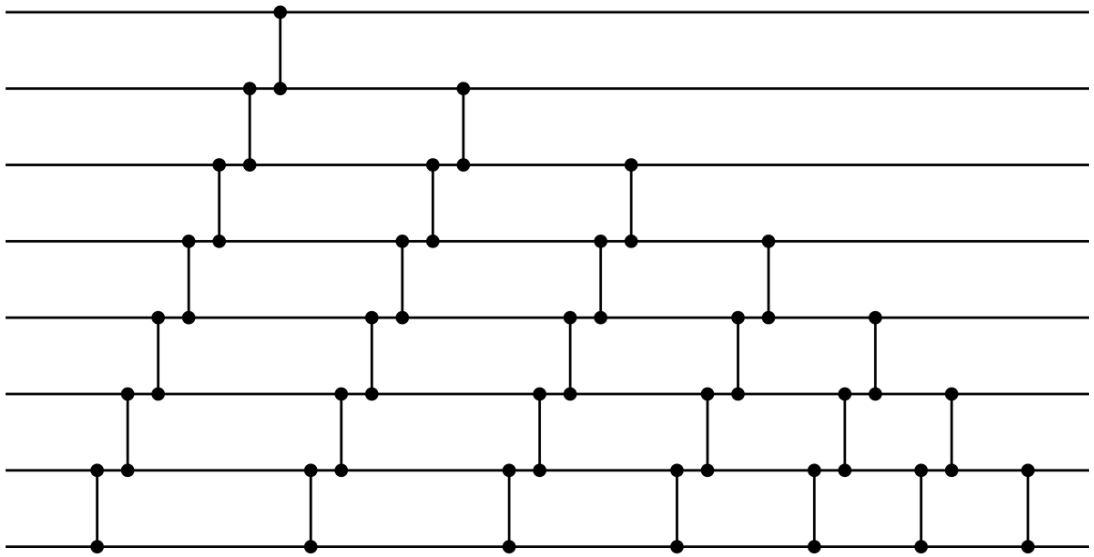
Sorting Networks

- Circuits for sorting fixed number n of inputs
  - Basic “gate” compare-and-swap:
    cmpswap(x, y) := (min(x, y), max (x, y))
  - Interesting challenge to get smallest sorting network
    for *n = 11* size only known to be between *33* and *35* compare-and-swap operations



- Zero-one principle
  - Correctness of sorting network (it sorts!) . . .
    . . . only requires sorting 0 and 1 inputs (bits) . . .
    . . . as long only compare-and-swap is used.
- Asymptotic complexity of algorithms
  - Examples: Bitonic Sorting, Batcher Odd-Even Merge Sort
  - with *span* = $O(\log^2 n)$
  - with *work* = $O(n \cdot \log^2 n) = T1$
  - but sequential time $T = O(n \cdot \log n)$
  - maximum absolute speed-up $S_n = O(n/ \log n)$

Bubble Sort Example



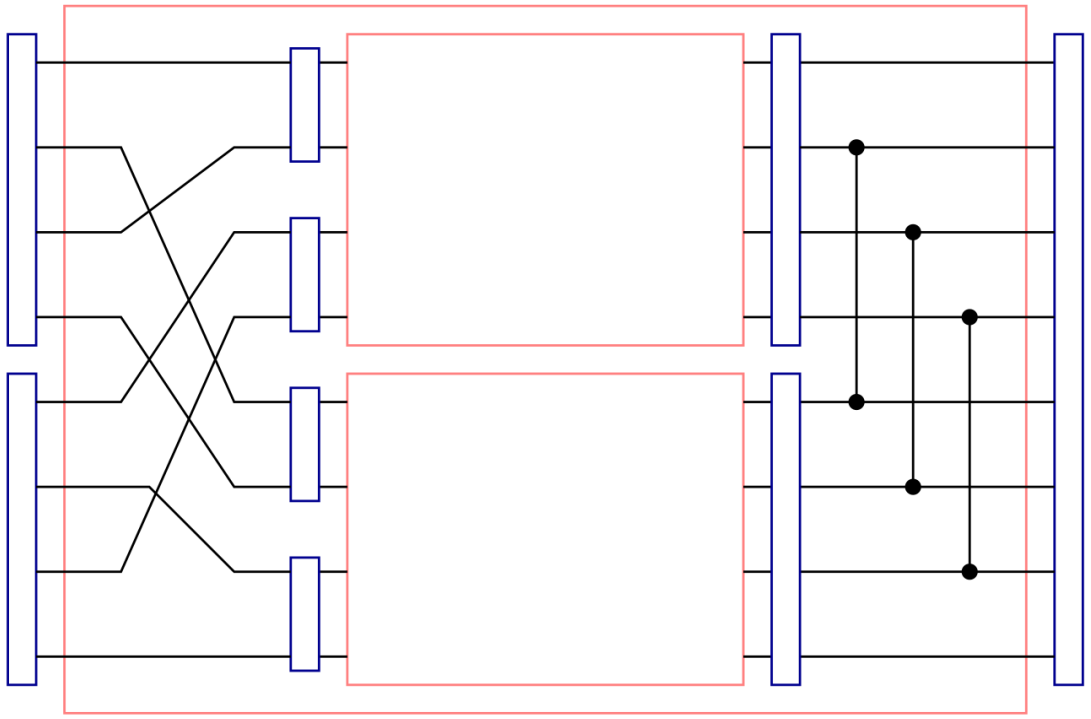
- Top-most *i* sorted after *i* phases
- Lowest value only sorted after *n − 1* compare-and-swaps
- *work* = $O(n^2)$
- *span* = $O(n)$
- Looks like perfect speedup $S_n = O(n)$ w.r.t. (bad) sequential algorithm
- However, if we compare against Quicksort

  $T = O(n \cdot \log n)$

  we only get

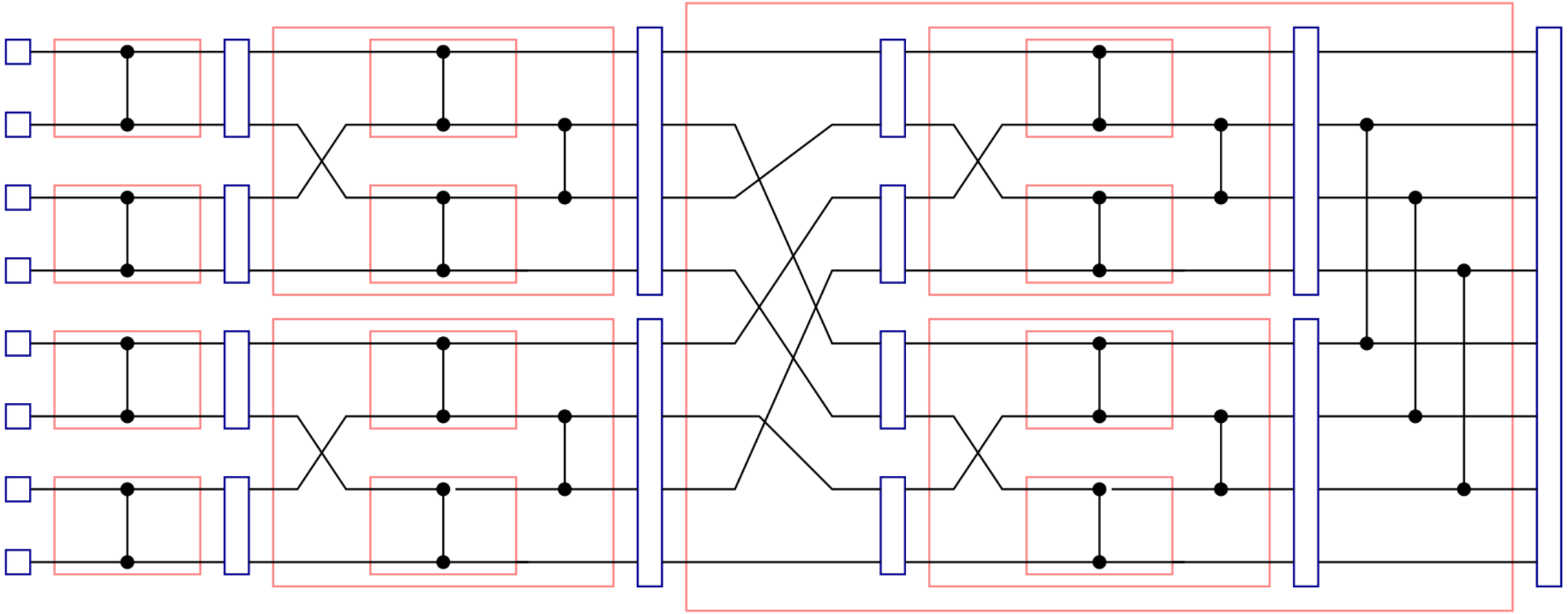
$$S_n = \mathcal{O}(\tfrac{n \cdot \log n}{n}) = \mathcal{O}(\log n) < \mathcal{O}(n/\log n)$$

Batcher Odd-Even Merge Sort



- Basically as merge sort
  - Split input into two parts . . .
    . . . sort parts recursively . . .
    . . . merge sorted sequence.
- Example: recursion for $n = 8$
  - Outer block takes two sorted sequences of size 4 each
  - Each inner block takes two sorted sequences of size 2 each
  - Outer input sequences need to be sorted too

Batcher Odd-Even Merge Sort



NC – Nick's Class

- f (n) polylogarithmic *iff* exists constant *c* such that $f(n) = O(\log^c n)$

- NC is set of decision problems . . .
  . . . which can be decided in polylogarithmic time . . .
  . . . on a parallel computer with polynomial many processors, i.e., . . .
  . . . exists constant *c* such that $p = O(n^k)$.

- NC^c requires (parallel) computation time (*span*) in $O(\log^c n)$

- $NC = \cup NC^c$

L, NL, AC

- **L** is set of decision problems solvable in logarithmic space deterministically
- **NL** is set of decision problems with logarithmic space non-deterministically
- **NC = AC** is the set of decision problems with logarithmic circuit complexity, i.e., . . .
  . . . each input of size n can be decided by polynomial circuit with logarithmic depth in n, . . .
  . . . made of gates with bounded (NC) or unbounded (AC) number of inputs
- As before define NC^c and AC^c requiring $O(\log^c n)$ depth (layers)

P Completeness

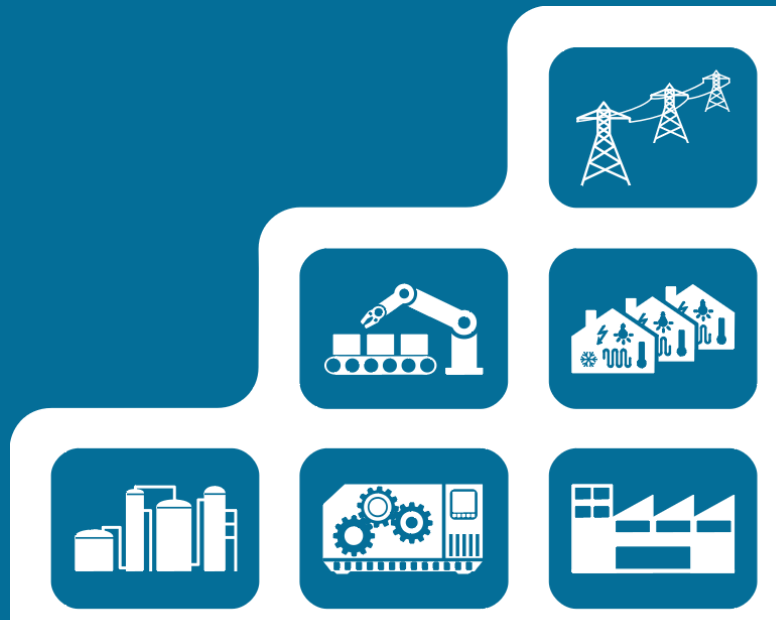
$$NC^1 \subseteq L \subseteq NL \subseteq AC^1 \subseteq NC^2 \subseteq AC^2 \subseteq NC^3 \subseteq \cdots \subseteq NC = AC \subseteq P$$

- Using “logarithmic” reductions
- It is commonly believed that NC ≠ P
- Accordingly P-hard problems are supposed to be NOT “parallelizable”
- Similar to the common belief that P ≠ NP

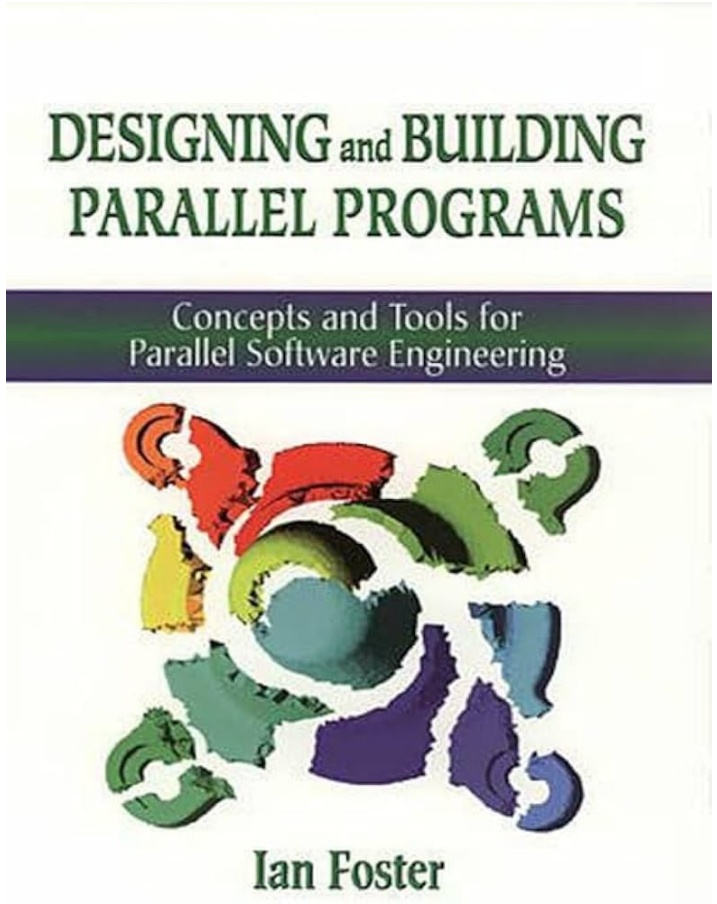
Circuit Evaluation Problem

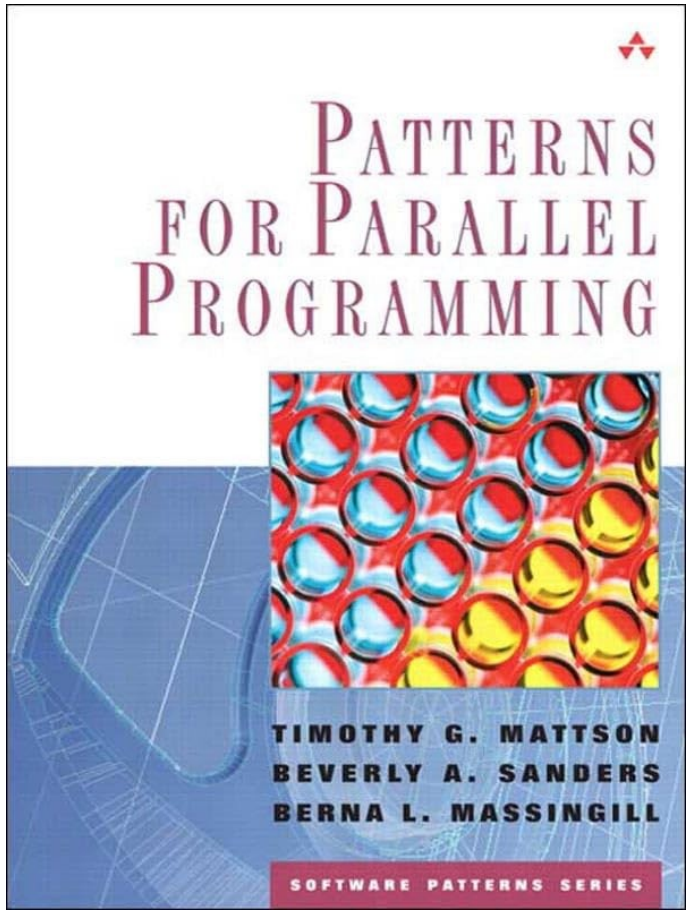
- Given a boolean circuit with one output, and an evaluation to its inputs.
- Evaluate the circuit and determine its output value for that input assignment.
- This problem (deciding whether output yields one) is P-complete . . .
  . . . and thus considered not to be parallelizable.
- Thus evaluating a function can not be done “effectively” in parallel.
- One step of simulation or constraint propagation are not parallelizable! (?)

Parallel Design Patterns

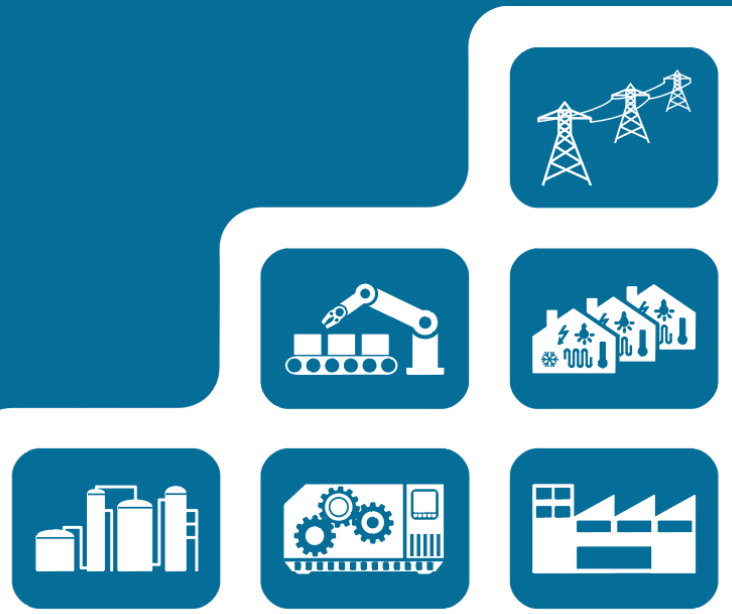


Guidelines and Methodologies for Implementing Parallel Programs

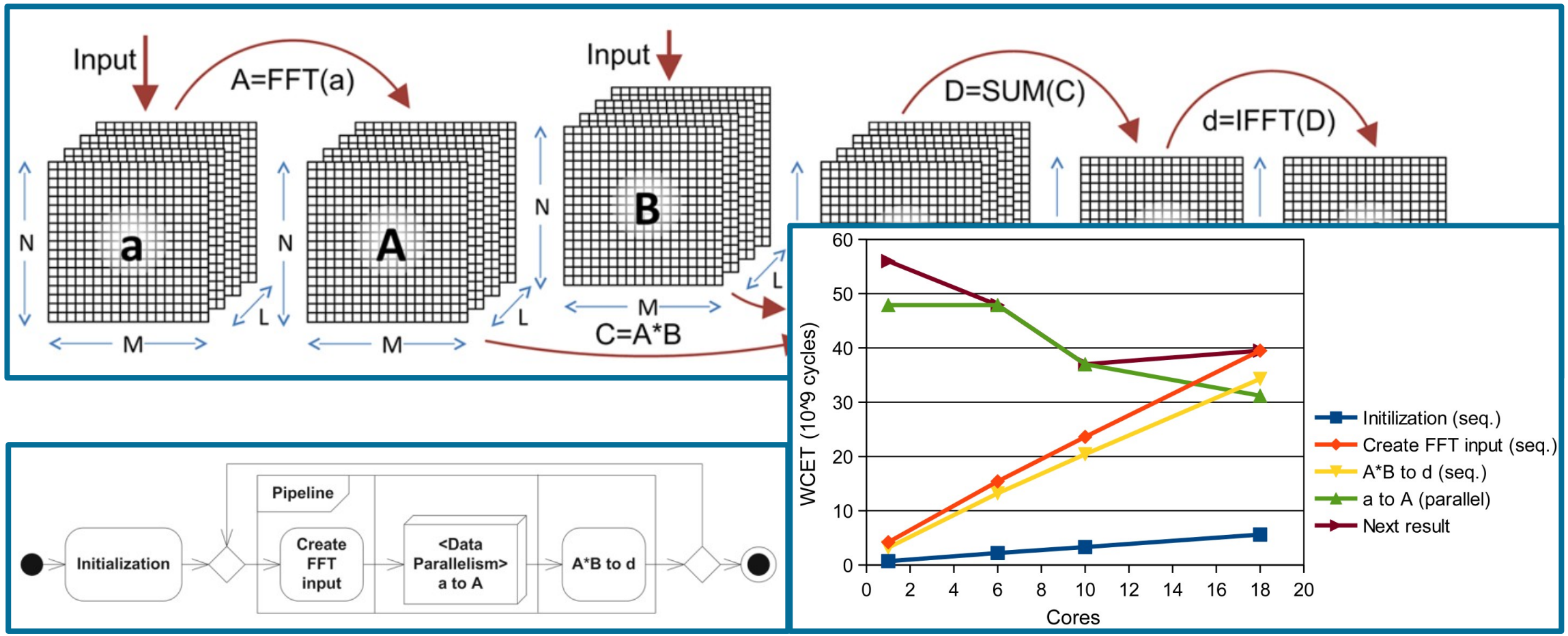




Examples from Real-Time Domain

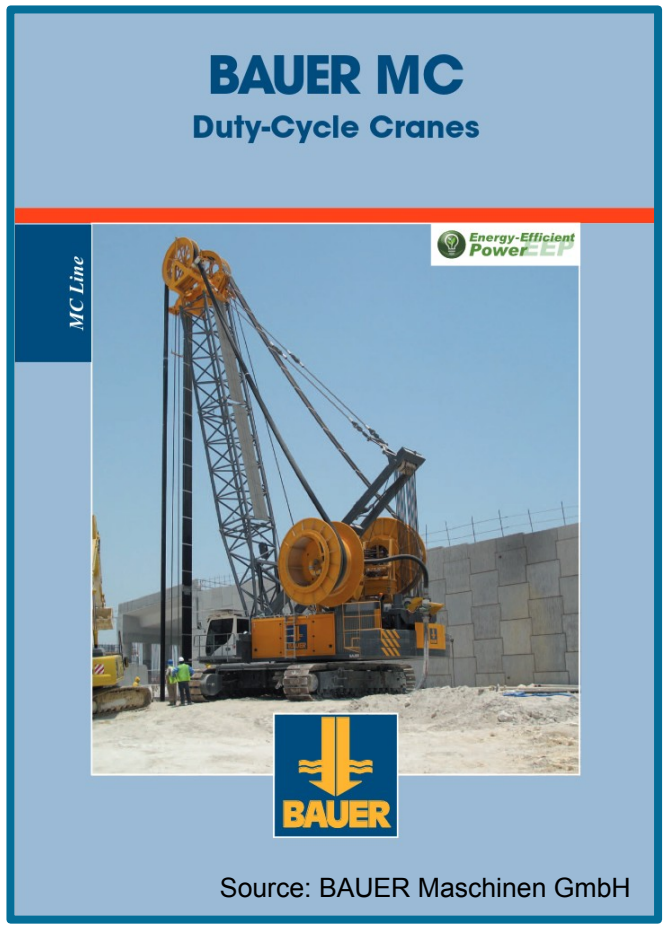


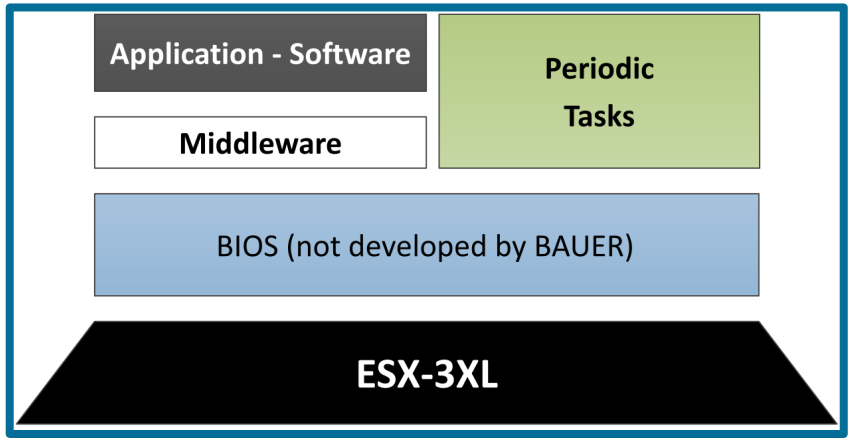
Signal Processing Pipeline

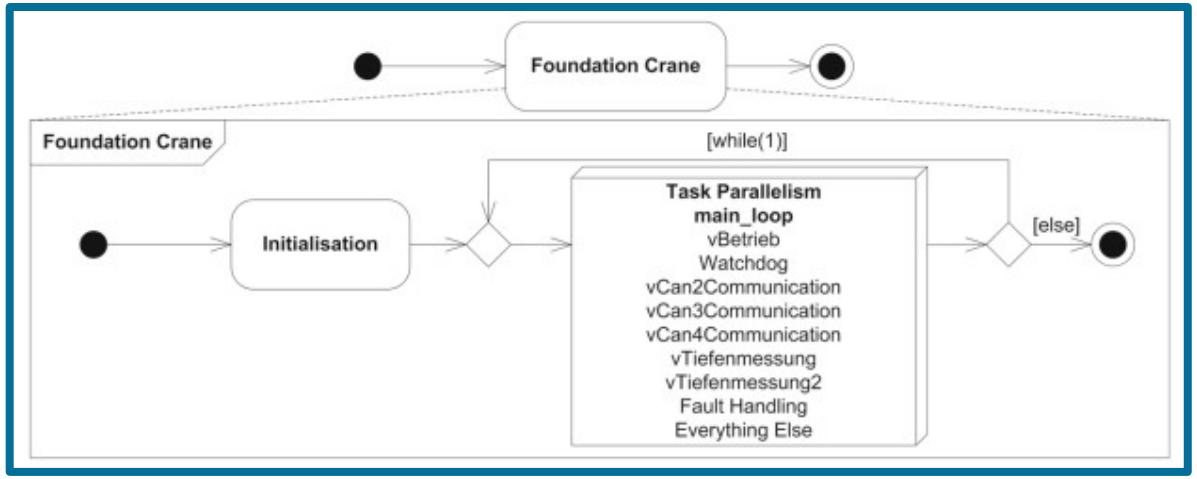


Frieb, M., Jahr, R., Ozaktas, H. et al. A Parallelization Approach for Hard Real-Time Systems and Its Application on Two Industrial Programs. Int J Parallel Prog 44, 1296–1336 (2016). https://doi.org/10.1007/s10766-016-0432-7

Control Program of BAUER MC 128







Frieb, M., Jahr, R., Ozaktas, H. et al. A Parallelization Approach for Hard Real-Time Systems and Its Application on Two Industrial Programs. Int J Parallel Prog 44, 1296–1336 (2016). https://doi.org/10.1007/s10766-016-0432-7

Control Program of BAUER MC 128

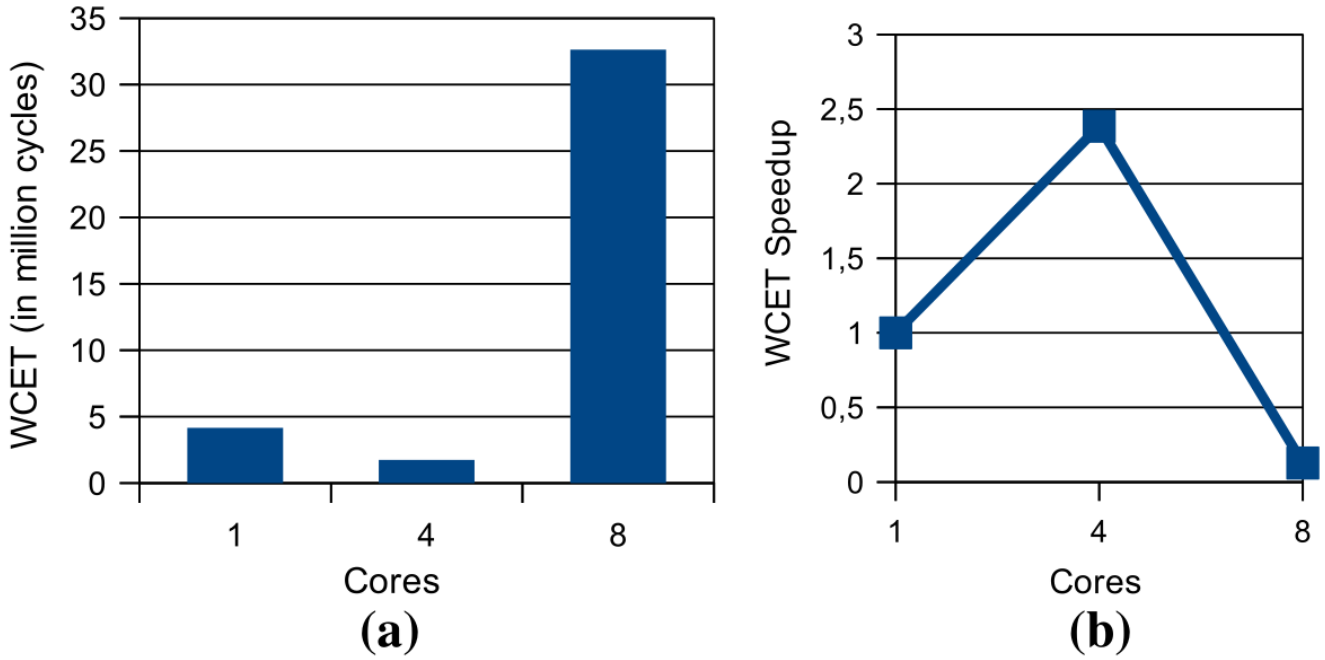


Fig. 17 Evaluation results: static WCETs and WCET speedup, **a** WCET for 1, 4 and 8 cores. At the 4 core version, the WCET falls to around 40 % of the sequential version, while it rises to a multiple at the 8 core version, **b** WCET Speedup for 1, 4 und 8 cores. At 4 cores, the speedup reaches around 2.4, while at 8 cores a slowdown to around 0.15 shows up

Frieb, M., Jahr, R., Ozaktas, H. et al. A Parallelization Approach for Hard Real-Time Systems and Its Application on Two Industrial Programs. Int J Parallel Prog 44, 1296–1336 (2016). https://doi.org/10.1007/s10766-016-0432-7

LINZ INSTITUTE
OF TECHNOLOGY

Thank you!

Univ.-Prof. Dr. Alois Zoitl, alois.zoitl@jku.at
LIT | Cyber-Physical Systems Lab
Johannes Kepler University Linz